Why Weighted Finite-State Transducers?

1. **Efficiency and Generality of Classical Automata Algorithms**
    - Efficient algorithms for a variety of problems (e.g. string-matching, compilers, Unix, design of controllability systems in aircrafts).
    - General algorithms: rational operations, intersection.
2. **Weights**
    - Handling uncertainty: text, handwritten text, speech, image, biological sequences.
    - Increased generality: finite-state transducers, multiplicity.
3. **Applications**
    - Text: pattern-matching, indexation, compression.
    - Speech: Large-vocabulary speech recognition, speech synthesis.
    - Image: image compression, filters.

Software Libraries

- **FSM Library**: Finite-State Machine Library – general software utilities for building, combining, optimizing, and searching weighted automata and transducers.
  `http://www.research.att.com/sw/tools/fsm/`
- **GRM Library**: Grammar Library – general software collection for constructing and modifying weighted automata and transducers representing grammars and statistical language models.
  `http://www.research.att.com/sw/tools/grm/`

Weight Sets: Semirings

A *semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ = a ring that may lack negation.

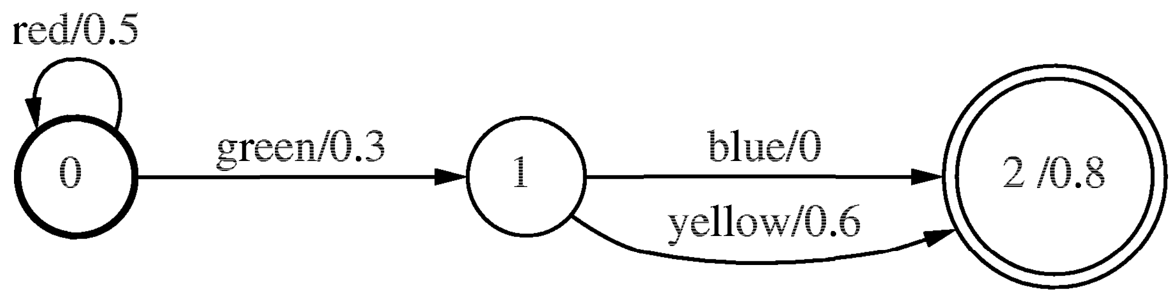
- **Sum**: to compute the weight of a sequence (sum of the weights of the paths labeled with that sequence).
- **Product:** to compute the weight of a path (product of the weights of constituent transitions).

Semiring	Set	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\min$	$+$	$+\infty$	0

with $\oplus_{\log}$ defined by: $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

Automata/Acceptors

- **Graphical Representation** (`A.ps`):



- **Acceptor File** (`A.txt`):

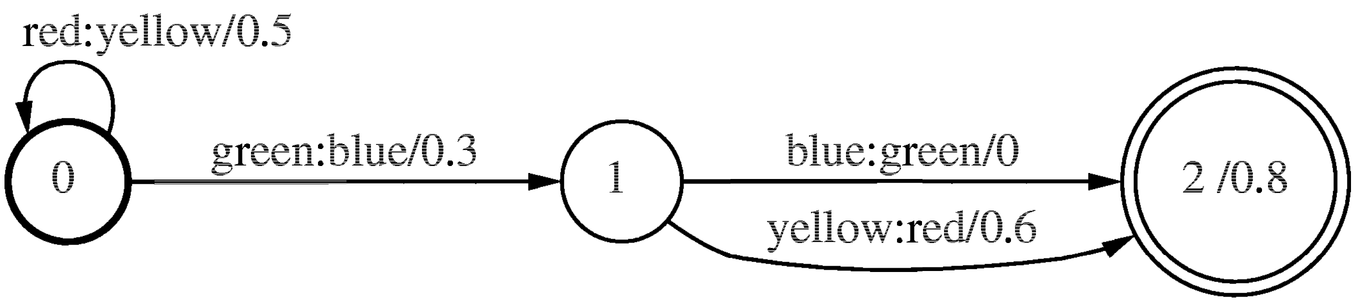
```
0   0   red      .5
0   1   green    .3
1   2   blue
1   2   yellow   .6
2   .8
```

- **Symbols File** (`A.syms`):

```
red      1
green    2
blue     3
yellow   4
```

Transducers

- **Graphical Representation** (`T.ps`):



- **Transducer File** (`T.txt`):

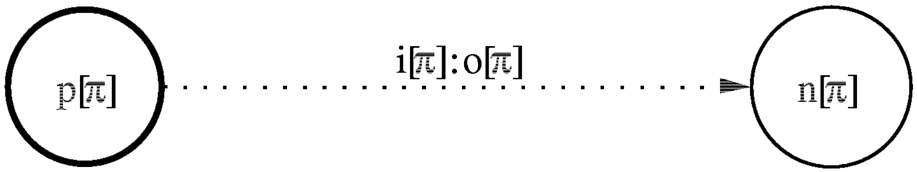
```
0   0   red     yellow  .5
0   1   green   blue    .3
1   2   blue    green
1   2   yellow  red     .6
2   .8
```

- **Symbols File** (`T.syms`):

```
red      1
green    2
blue     3
yellow   4
```

Definitions and Notation – Paths

- **Path π**
  - Origin or previous state: $p[\pi]$.
  - Destination or next state: $n[\pi]$.
  - Input label: $i[\pi]$.
  - Output label: $o[\pi]$.



- **Sets of paths**
  - $P(R_1, R_2)$: set of all paths from $R_1 \subseteq Q$ to $R_2 \subseteq Q$.
  - $P(R_1, x, R_2)$: paths in $P(R_1, R_2)$ with input label x.
  - $P(R_1, x, y, R_2)$: paths in $P(R_1, x, R_2)$ with output label y.

Definitions and Notation – Automata and Transducers

1. **General Definitions**
   - Alphabets: input Σ, output Δ.
   - States: Q, initial states I, final states F.
   - Transitions: $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$.
   - Weight functions:

     initial weight function $\lambda : I \rightarrow \mathbb{K}$

     final weight function $\rho : F \rightarrow \mathbb{K}$.

2. **Machines**
   - Automaton $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ with for all $x \in \Sigma^*$:

$$[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

   - Transducer $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ with for all $x \in \Sigma^*, y \in \Delta^*$:

$$[\![T]\!](x, y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

Rational Operations – Algorithms

- **Definitions**

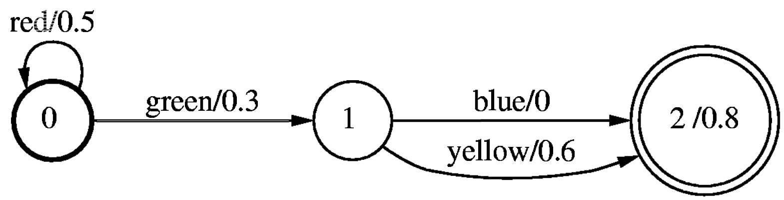
Operation	Definition and Notation
Sum	$[\![T_1 \oplus T_2]\!](x,y) = [\![T_1]\!](x,y) \oplus [\![T_2]\!](x,y)$
Product	$[\![T_1 \otimes T_2]\!](x,y) = \bigoplus_{x=x_1 x_2, y=y_1 y_2} [\![T_1]\!](x_1,y_1) \otimes [\![T_2]\!](x_2,y_2)$
Closure	$[\![T^*]\!](x,y) = \bigoplus_{n=0}^{\infty} [\![T]\!]^n(x,y)$

- **Conditions on the closure operation**: condition on T: e.g. weight of ϵ-cycles $= \overline{0}$ (*regulated transducers*), or semiring condition: e.g. $\overline{1} \oplus x = \overline{1}$ as with the tropical semiring (*locally closed semirings*).

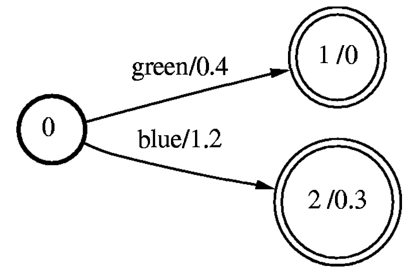
- **Complexity and implementation**
  - Complexity (linear): $O((|E_1| + |Q_1|) + (|E_2| + |Q_2|))$ or $O(|Q| + |E|)$.
  - Lazy implementation.

Sum – Illustration

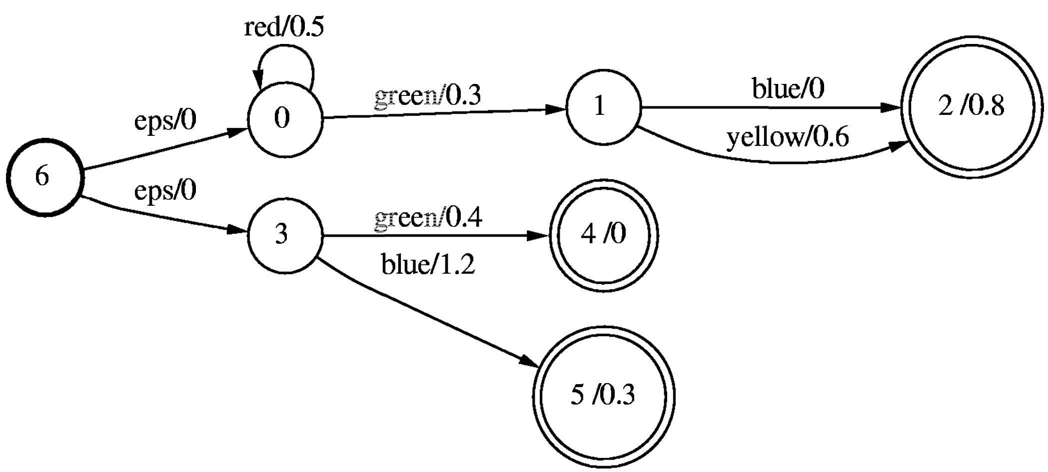
- **Program:** `fsmunion A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



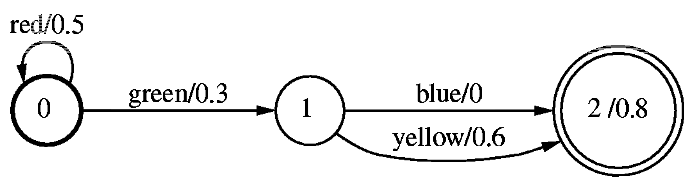
B.fsa



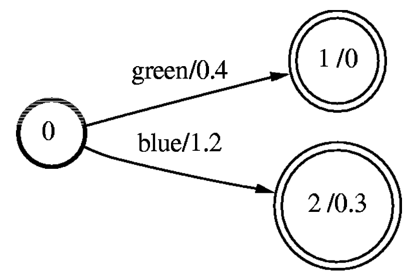
C.fsa

Product – Illustration

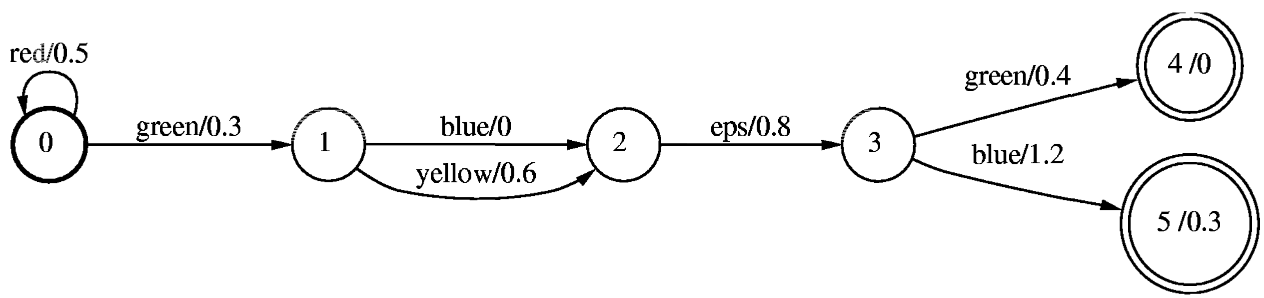
- **Program:** `fsmconcat A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



B.fsa



C.fsa

Some Elementary Unary Operations – Algorithms

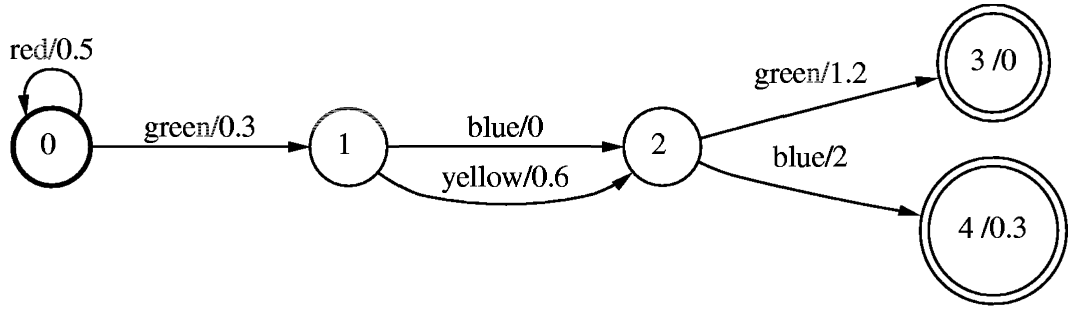
- **Definitions**

Operation	Definition and Notation	Lazy Implementation
Reversal	$[\![\widetilde{T}]\!](x, y) = [\![T]\!](\widetilde{x}, \widetilde{y})$	No
Inversion	$[\![T^{-1}]\!](x, y) = [\![T]\!](y, x)$	Yes
Projection	$[\![A]\!](x) = \bigoplus_{y} [\![T]\!](x, y)$	Yes

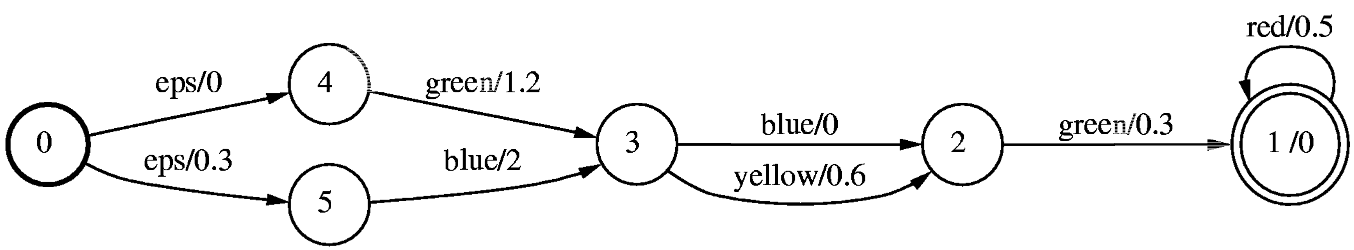
- **Complexity and implementation**
  - Complexity (linear): $O(\|Q\| + \|E\|)$.
  - Lazy implementation (see table).

Reversal – Illustration

- **Program:** `fsmreverse A.fsm >C.fsm`
- **Graphical Representation:**



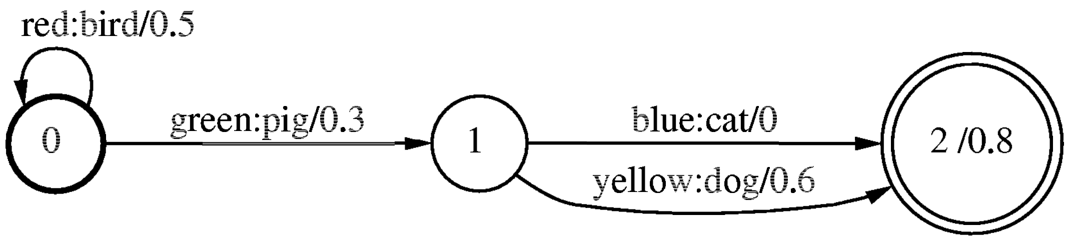
A.fsa



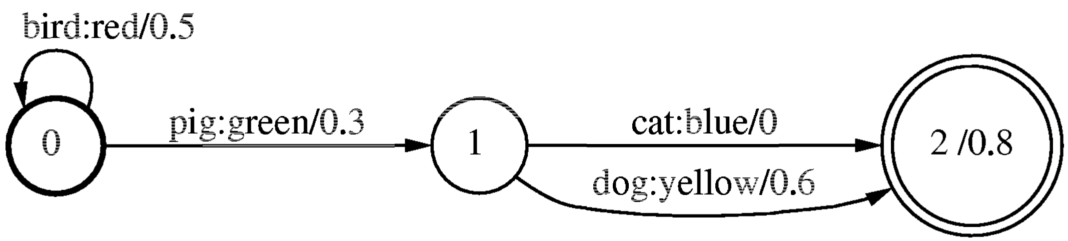
C.fsa

Inversion – Illustration

- **Program:** `fsminvert A.fsm >C.fsm`
- **Graphical Representation:**



A.fst



C.fst

Some Fundamental Binary Operations – Algorithms

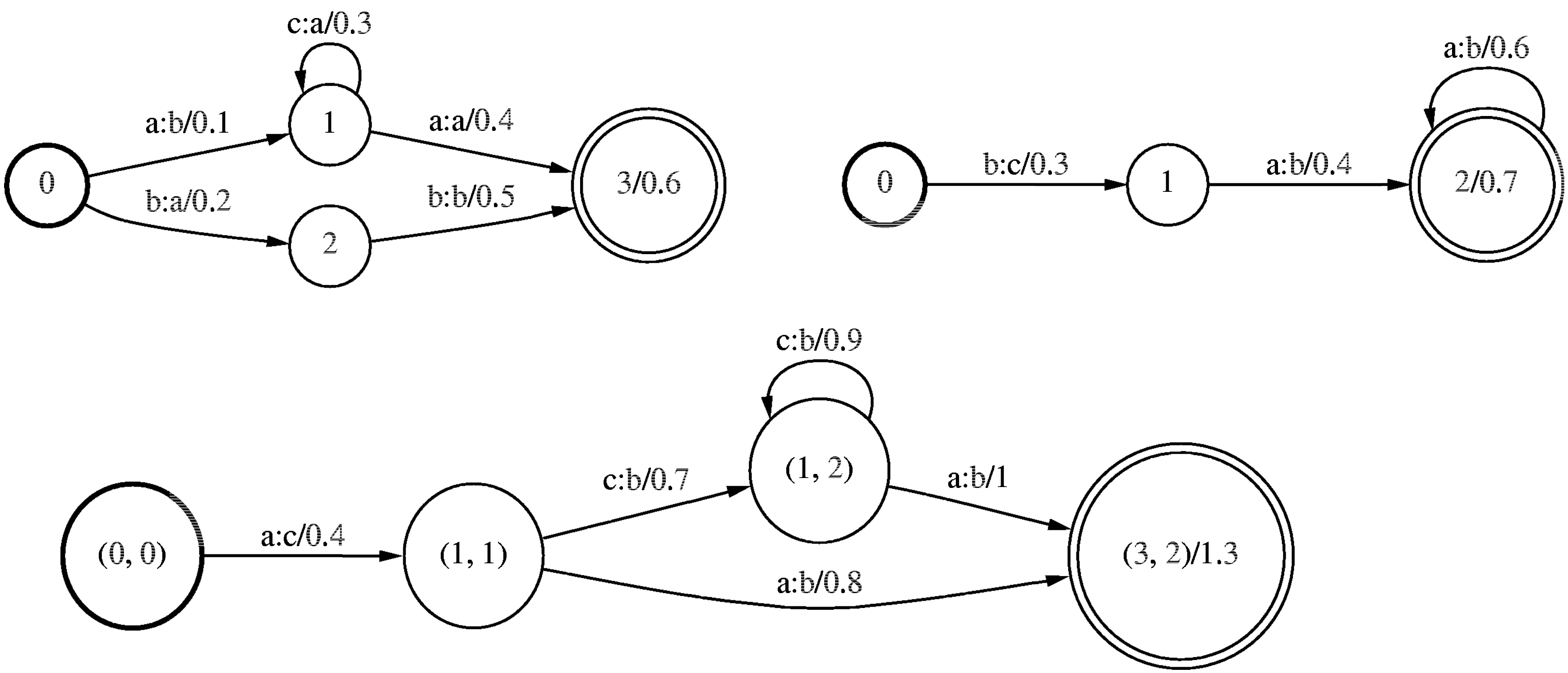
- **Definitions**

Operation	Definition and Notation	Condition
Composition	$[\![T_1 \circ T_2]\!](x, y) = \bigoplus_z [\![T_1]\!](x, z) \otimes [\![T_2]\!](z, y)$	$\mathbb{K}$ commutative
Intersection	$[\![A_1 \cap A_2]\!](x) = [\![A_1]\!](x) \otimes [\![A_2]\!](x)$	$\mathbb{K}$ commutative
Difference	$[\![A_1 - A_2]\!](x) = [\![A_1 \cap \overline{A_2}]\!](x)$	A_2 unweighted & deterministic

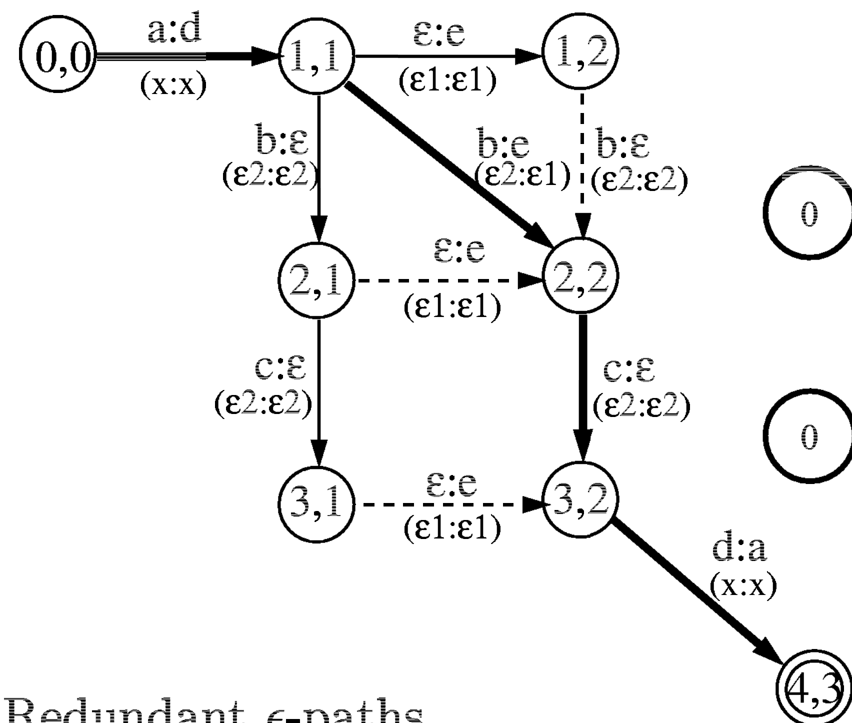
- **Complexity and implementation**
  - Complexity (quadratic): $O((|E_1| + |Q_1|)\,(|E_2| + |Q_2|))$.
  - Path multiplicity in presence of ϵ-transitions: ϵ-filter.
  - Lazy implementation.

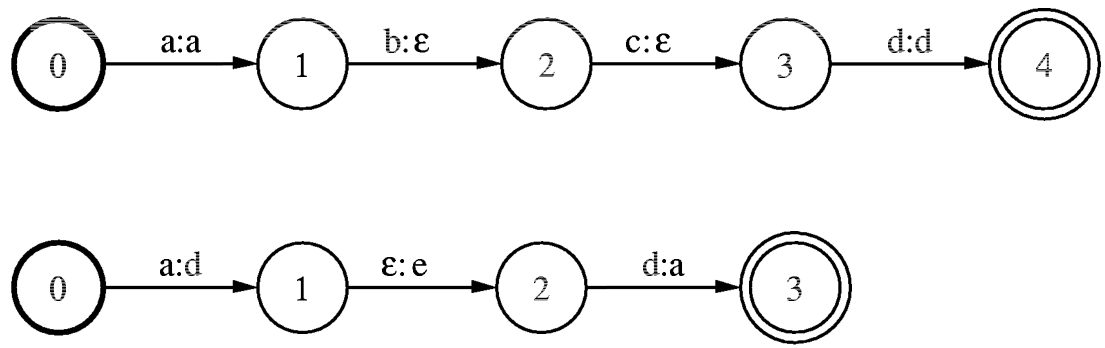
Composition – Illustration

- **Program:** `fsmcompose A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



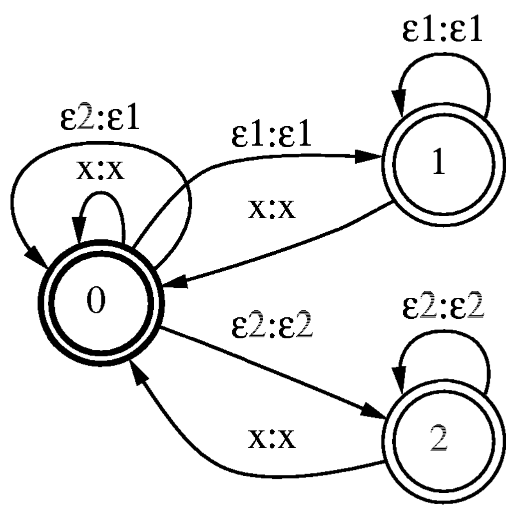
Multiplicity & ϵ-Transitions – Problem





Redundant ϵ-paths.

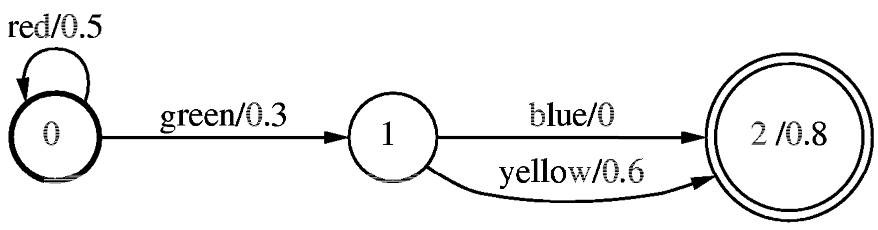
Solution – Filter F for Composition



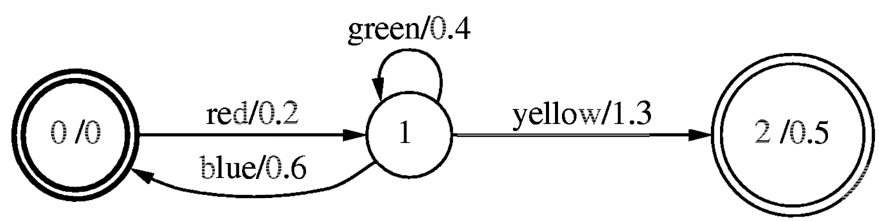
Replace $T_1 \circ T_2$ by $T_1 \circ F \circ T_2$.

Intersection – Illustration

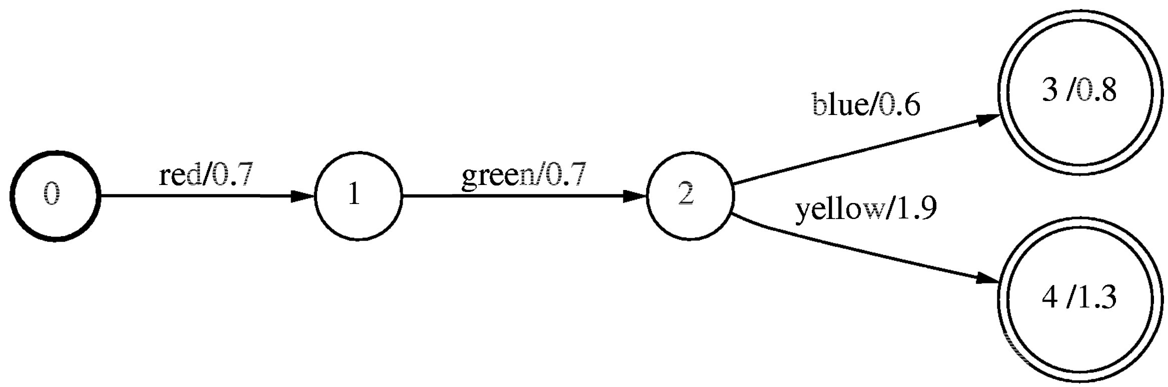
- **Program:** `fsmintersect A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



B.fsa



C.fsa

Single-Source Shortest-Distance Algorithms – Algorithm

- **Generic single-source shortest-distance algorithm**
  - Definition: for each state q,

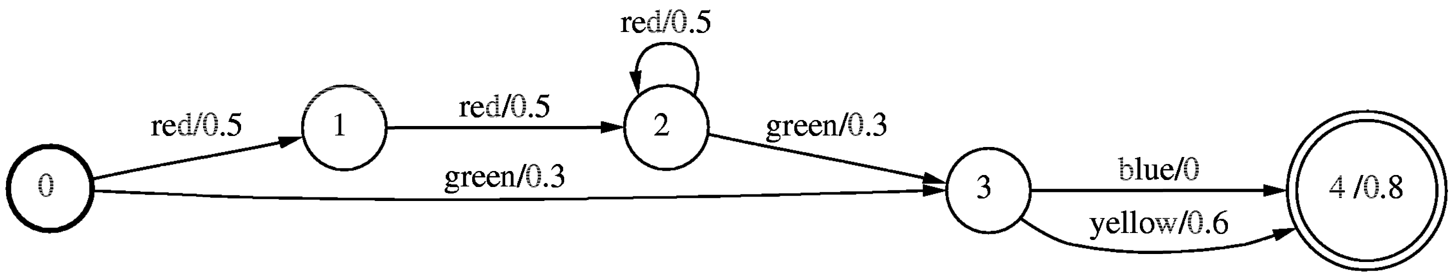
$$d[q] = \bigoplus_{\pi \in P(q,F)} w[\pi]$$

  - Works with any queue discipline and any semiring k-closed for the graph.
  - Coincides with classical algorithms in the specific case of the tropical semiring and the specific queue disciplines: best-first (Dijkstra), FIFO (Bellman-Ford), or topological sort order (Lawler).

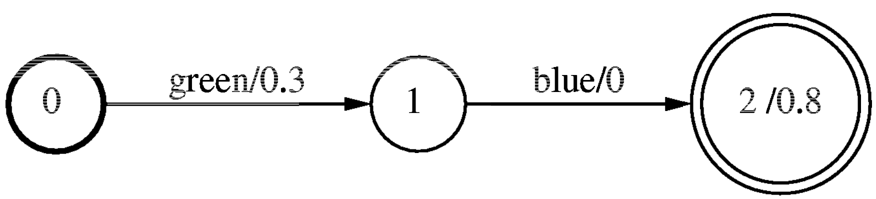
- ***N*-best strings algorithm**
  - General N-best paths algorithm augmented with the computation of the potentials.
  - On-the-fly weighted determinization.

Single-Source Shortest-Distance Algorithms – Illustration

- **Program:** `fsmbestpath [-n N] A.fsm >C.fsm`
- **Graphical Representation:**



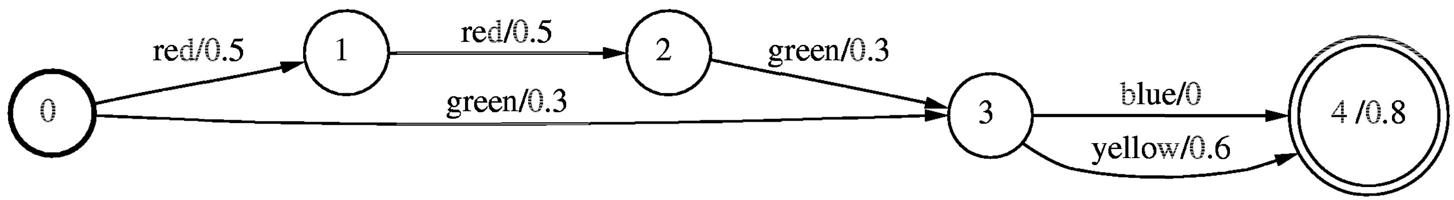
A.fsa



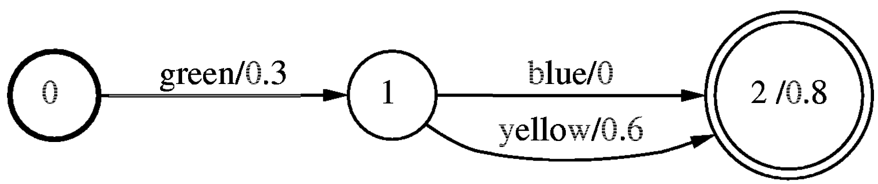
C.fsa

Pruning – Illustration

- **Program:** `fsmprune -c1.0 A.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



C.fsa